ASP

Answer Set Programming

Kelemen Attila
mrdiablox@gmail.com

Kedvcsináló
N királynő 3+1 sorban

```
index(1..n).

% minden sorban pontosan 1 királynő van
1{q(X,Y):index(X)}1 :- index(Y).

% az rossz, ha ugyanabban az oszlopban 2 királynő van
:- index(X; Y1; Y2), q(X, Y1), q(X, Y2), Y1 != Y2.

% az rossz, ha egy átlóban 2 királynő van
:- index(X1; X2; Y1; Y2), q(X1, Y1), q(X2, Y2), Y1 != Y2,
                                        abs(X1 - X2) == abs(Y1 - Y2).
```

Mi az ASP?

Prolog	ASP
Elsőrendű logika / Horn-klózok	Ítéletlogika / Grounding
Következik az állítás a programból?	Létezik-e modellje a programnak?
A válasz változó kötések	A válasz a program modellje

Definíciók

A megengedett szabályok:

$p_0 \leftarrow p_1, \ldots, p_m, \text{not } p_{m+1}, \ldots, \text{not } p_n$, ahol $n \geq m \geq 0$, és minden p_i egy atom, és a „not” jelentése: nem bizonyítható.

Legyen egy ilyen szabály r, ekkor

- fej(r) = p_0
- törzs(r) = $\{p_1, \ldots, p_m, \text{not } p_{m+1}, \ldots, \text{not } p_n\}$
- törzs$^+$(r) = $\{p_1, \ldots, p_m\}$
- törzs$^-$(r) = $\{\text{not } p_{m+1}, \ldots, \text{not } p_n\}$

És megint definíciók . . .

- Egy Π program egyszerű, ha $\forall$ r $\in \Pi \rightarrow$ törzs$^-$(r) = $\varnothing$
- X atomok halmaza zárt az egyszerű Π programon,
  ha $\forall$ r $\in \Pi \rightarrow$ (törzs$^+$(r) $\subseteq$ X $\rightarrow$ fej(r) $\in$ X)
- A legkisebb ilyen zárt halmazt válaszhalmaznak nevezzük.
  Ez fogja nekünk adni a modell leírását. Jelölésben: Cn(Π).
- Π^X = { fej(r) $\leftarrow$ törzs$^+$(r) | r $\in \Pi$, törzs$^-$(r) $\cap$ X = $\varnothing$ }: Π program redukáltja X felett
- Egy Π program válaszhalmaza X = Cn(Π^X)

Ennyi ☺

Írjunk programot!

Legyen egy példa a Π programra {p ← not q, q ← not p}

X	Π^X	$Cn(\Pi^X)$	
∅	p ← q ←	{p, q}	
{p}	p ←	{p}	✓
{q}	q ←	{q}	✓
{p, q}		∅	

Elképzelhető, hogy mi nem szeretjük a „q”-t, ezért szeretnénk, ha ez nem lenne megoldás.

Nézzük először a {a ← not a} programot, aminek nincs válaszhalmaza.

- Miért?
- Mert ha belevesszük, a válaszhalmazba, a „not a” megakadályozza, hogy „a”-t levezessük, ha nem akkor pedig le tudnánk vezetni.

Használjuk ezt ki, és vezessünk be a programunkba egy új „f” atomot és egy új szabályt: f ← not f, q

Ez megakadályozza, hogy q-t levezessük.

Továbbiakban:
„f ←not f, p_1, . . . , p_m, not p_{m+1}, . . . , not p_n”, helyett
„← p_1, . . . , p_m, not p_{m+1}, . . . , not p_n”

Most már bármilyen programot megírhatunk ☺

Egy egyszerű stratégia a programunk írásra a jól ismert „Generate and Test”.

1. Írjunk olyan szabályokat, amik leírják, hogy milyen lehet egy modell
2. Írjunk olyan szabályokat, amik leírják, hogy milyen nem lehet egy modell

Az 1. lépést általában egyszerűen leírhatjuk.
A 2. lépésben „$\leftarrow$ A” alakú szabályokkal tilthatunk le 1. szerint jó modelleket.

Változók ítéletkalkulusban? Grounding

Nincsenek, de:

Tegyük fel, hogy a predikátum-kalkulusbeli formulákban a változóinknak véges az értékkészlete.

Vegyük példának a következőket:

1. vilagit(lampa).
2. vilagit(nap).
3. $\forall$x fenyes(x) $\leftarrow$ vilagit(x).

$\Rightarrow$

fenyes(lampa) $\leftarrow$ vilagit(lampa) $\wedge$
fenyes(nap) $\leftarrow$ vilagit(nap) $\wedge$
fenyes(feketelyuk) $\leftarrow$ vilagit(feketelyuk) $\wedge$ ...

Látszik, hogy X értékei, amikor az implikáció nem triviális: {lampa, nap}, azaz a tényállításokból ki tudjuk találni.

Így át tudjuk alakítani a 3-t két ítéletlogikai szabállyá.

Lparse: egy grounder

Feladata: A változók eliminálása, ezzel ítéletlogikai alakra hozni a problémát.

Amit még megtesz:

- Klasszikus tagadás értelmezése (később)
- Változókon függvények elvégzése (abs, kivonás, összehasonlítás ...)
- Egyszerűsítések

Példa: Bűvös négyzet

Definíció: Egy NxN-s mátrix bűvös négyzet, ha nincs olyan sor vagy oszlop melyben van két ugyanolyan elem. Elemei pozitív egészek, és N-nél nem nagyobbak. (+kikötés: az 1. sor x. oszlopában lévő elem: x)

Első próba:

```
index(1..n).

e(X, 1, X) :- index(X).
e(X, Y, K) :- index(X; Y; K).

:- index(X; Y1; Y2; K), e(X, Y1, K), e(X, Y2, K), Y1 != Y2.
:- index(X1; X2; Y; K), e(X1, Y, K), e(X2, Y, K), X1 != X2.
```

Mi a hiba?

Javítás: Bűvös négyzet

Logikai hiba: **Semmi sem gátolta meg, hogy 1 pozíción több elem legyen.**
Szemantikai hiba: **Mindent tényként közöltünk, és így nem tudtunk kizárni.**

```
index(1..n).

e(X, 1, X) :- index(X).

e(X, Y, K) :- index(X; Y; K), not -e(X, Y, K).
-e(X, Y, K) :- index(X; Y; K; N), e(X, Y, N), N != K.

:- index(X; Y1; Y2; K), e(X, Y1, K), e(X, Y2, K), Y1 != Y2.
:- index(X1; X2; Y; K), e(X1, Y, K), e(X2, Y, K), X1 != X2.
```

Ahol a „-” a klasszikus tagadást jelöli.

Klasszikus tagadás

Mi volt a problémánk az előző példánál?

- Ki szerettük volna fejezni, hogy valami nincs a modellben, és erre a meghiúsulásos tagadás kevés volt.

Elvileg A és ¬A – t is tekinthetjük külön szimbólumnak.
A probléma csak akkor jelenik meg, ha A és ¬A is megjelenne a válaszhalmazban.

Erre a matematika válasza:
- Ebben a világaban minden állítás igaz. Gondoljunk a „Hamis $\rightarrow$ A” – ra.

Tehát a válaszhalmaz minden literált kell, hogy tartalmazzon.

Ezért ilyen szabályok kellenek (a ‘-k a szimbólum negáltját jelölik):

$q \leftarrow p, p'$ $p \leftarrow q, q'$
$q' \leftarrow p, p'$ $p \leftarrow q, q'$ ...
...

A gyakorlatban nem tűnik a legjobb ötletnek, ha egy minden literált tartalmazó halmazt adunk vissza, hiszen általában ez a halmaz nem használható.

Gyakran jobb, ha ezt nem fogadjuk el válaszhalmaznak.
Ezt a következő szabályokkal tehetjük meg:
← p, p'
← q, q'
...

Az Lparse opcionálisan lehetőséget biztosít az előbbi mechanizmusokat felhasználva a klasszikus negációra, és mindkettő filozófiát megengedi.
Az alapértelmezett az utóbbi.

megj.: Valószínűleg nem tökéletes az algoritmusa, mert sikerült klasszikus negációval végtelenciklusba juttatnom ☺.

Kardinalitás

Alakja: $K = \min \{q_1, ..., q_n\}$ max. ahol $n \geq 1$

Jelentése: X válaszhalmaz kielégíti K-t, ha $\min \leq |X \cap K| \leq \max$

Magyarul: X halmazban legalább „min” darab, legfeljebb „max” darab eleme van K-nak

Egy kis kiegészítés: $\min \{q_1, ..., q_n : f\}$ max
Itt f egy feltételes literál, f-nek mindenképpen igaznak kell lennie.
(q-k közül lehet vele szűrni)

Komplexitás: NP - teljes

Bűvös négyzet kardinális felhasználásával

```
index(1..n).

e(X, 1, X) :- index(X).

% minden (X, Y) pozíción pontosan egy féle elem van
1{e(X, Y, K):index(K)}1 :- index(X; Y).

:- index(X; Y1; Y2; K), e(X, Y1, K), e(X, Y2, K), Y1 != Y2.
:- index(X1; X2; Y; K), e(X1, Y, K), e(X2, Y, K), X1 != X2.
```

Az smodels

Egy implementáció a válaszhalmaz számítására íteletlogikában.

Az smodels algoritmusa

smodels(L, U)
1 expand(L,U)
2 if L = U then exit with L
3 if L $\not\subset$ U then return fail
4 4 A $\leftarrow$ select(U \ L)
5 smodels(L $\cup$ {A},U)
6 smodels(L,U \ {A})

expand(L, U)
repeat
L' := L
L := L $\cup$ answerset(Π^U)
if not (L $\subseteq$ U) then return
U' := U
U := U $\cap$ answerset(Π^L)
if not (L $\subseteq$ U) then return
until L = L' and U = U'

Példa: pelda_ember.txt

Preferenciák

Szabály preferencia:

Egy prioritást határozunk meg a szabályok között, és először mindig a nagyobb prioritású szabályt nézzük.

Rendezett diszjunkció:

A fej részben több atomot sorolunk fel, de ezt nem rendes diszjunkcióként értelmezzük. Legyen most p; q a fejben. Ez azt jelenti, hogy p-t gondoljuk oda csak, de ha az derülne ki, hogy p lehetetlen, akkor q-t.

Még több és komplexebb preferenciát tud leírni a PDL (preference description language).

Amiért jó ...

- Turing-teljes ☺
- Egyszerű, tisztán logikai leírás.
- Feketedoboz-szerű működés.
- A probléma triviális leírása esetén is meglepően gyorsan kapunk megoldást.

Amiért nem ...

- Nagy állapottér (pl.: nagy értékkészlet) esetén rengeteg atom, ezért lassú futás.
- Nem logikai jellegű feladatokkat nehéz, vagy közel lehetetlen vele megoldani.
- A kimenet általában nehezen olvasható, ezért többnyire csak valamilyen rendszerbe integrálva jó használni.

Eddigi felhasználási területei

- N királynő és bűvös négyzet ☺
- Orvosi diagnosztika
- Tervezés
- Termék konfiguráció
- Űrhajó reakcióvezérlése
- Kriptoanalízis
- Régi nyelvek tanulmányozása
- és még sok más ...